A. Key paper in which BBN described and defended its technical design

#13

Nat'l Computer Conf. 1975

Issues in packet switching network design*

by W. R. CROWTHER, F. E. HEART, A. A. McKENZIE, J. M. McQUILLAN, and D. C. WALDEN
Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

INTRODUCTION

The goals of this paper are to identify several of the key design choices that must be made in specifying a packet-switching network and to provide some insight in each area. Through our involvement in the design, evolution, and operation of the ARPA Network over the last five years (and our consulting in the design of several other networks), we have learned to appreciate both the opportunities and the hazards of this new technical domain.

The last year or so has seen a sudden increase in the number of packet-switching networks under consideration worldwide. It is natural that these networks try to improve on the example of the ARPA Network, and therefore that they contain many features different from those of the ARPA Network. We recognize that networks must be designed differently to meet different requirements; nevertheless, we think that it is easy to overlook important aspects of performance, reliability, or cost. It is vital that these issues be adequately understood in the development of very large practical networks—common user systems for hundreds or thousands of Hosts—since the penalties for error are correspondingly great.

Some brief definitions are needed to isolate the kind of computer network under consideration here:

Nodes. The nodes of the network are real-time computers, with limited storage and processing resources, which perform the basic packet-switching functions.

Hosts. The Hosts of the network are the computers, connected to nodes, which are the providers and users of the network services.

Lines. The lines of the network are some type of communications circuit of relatively high bandwidth and reasonably low error rate.

Connectivity. We assume a general. distributed topology in which each node can have multiple paths to other nodes, but not necessarily to all other nodes. Simple networks such as stars or rings are degenerate cases of the general topology we consider.

Message. The unit of data exchanged between source Host and destination Host.

Packet. The unit of data exchanged between adjacent nodes.

Acknowledgment. A piece of control information returned to a source to indicate successful receipt of a packet or message. A packet acknowledgment may be returned from an adjacent node to indicate successful receipt of a packet; a message acknowledgment may be returned from the destination to the source to indicate successful receipt of a message.

Store and Forward Subnetwork. The node stores a copy of a packet when it receives one, forwards it to an adjacent node, and discards its copy only on receipt of an acknowledgment from the adjacent node, a total storage interval of much less than a second.

Packet Switching. The nodes forward packets from many sources to many destinations along the same line, multiplexing the use of the line at a high rate.

Routing Algorithm. The procedure which the nodes use to determine which of the several possible paths through the network will be taken by a packet.

Node-Node Transmission Procedures. The set of procedures governing the flow of packets between adjacent nodes.

Source-Destination Transmission Procedures. The set of procedures governing the flow of messages between source node and destination node.

Host-Node Transmission Procedures. The set of procedures governing the flow of information between a Host and the node to which that Host is directly connected.

Host-Host Transmission Procedures. The set of procedures governing the flow of information between the source Host and the destination Host.

Within the class of network under consideration, there are already several operational networks and many network designs. The ARPA Network[1] is made up of over fifty node computers called IMPs and over seventy Hosts. The Cyclades Network[2] is a French network consisting of about six nodes and about two Hosts per node. The Societe Internationale de Telecommunication Aeronautique (SITA) Network[3] connects centers in eight or so cities mostly in Europe. The European Informatics Network (EIN),[4] also known as Cost-11, is currently in a design stage and will be a network interconnecting about six computers in several Common Market countries. Some other packet-switching network designs include: Autodin II,[5] NPL,[6] PCI,[7] RCP,[8] and Telenet.[7]

* This work was supported under Advanced Research Projects Agency Contracts DAHC15-69-C-0179 and F08606-73-C-0027.

Some of the more obvious differences among these networks can be cited briefly. The ARPA Network splits messages into packets up to 1000 bits long; the other networks have 2000-bit packets and no multipacket messages. Hosts connect to a single node in the ARPA Network and SITA; multiple connections are possible in Cyclades and EIN. Dynamic routing is used in the ARPA Network and EIN; a different adaptive method is used in SITA; fixed routing is presently used in Cyclades. The ARPA Network delivers messages to the destination Host in the same sequence as it accepts them from the source Host; Cyclades does not; in EIN it is optional. Clearly, many of the design choices made in these networks are in conflict with each other. The resolution of these conflicts is essential if balanced, high-performance networks are to be planned and built, particularly since many future designs will be intended for larger, less experimental, and more complex networks.

FUNDAMENTAL ISSUES

In this section we define what we believe are fundamental properties and requirements of packet-switching networks and what we believe are the fundamental criteria for measuring network performance.

Network properties and requirements

We begin by giving the properties central to packet-switching network design. The key assumption here is that the *packet processing algorithms* (acknowledgment/retransmission strategies used to control transmission over noisy circuits, routing, etc.) result in a virtual network path between the Hosts with the following characteristics:

a. Finite, fluctuating delay—A result of the basic line bandwidth, speed of light delays, queueing in the nodes, line errors, etc.
b. Finite, fluctuating bandwidth—A result of network overhead, line errors, use of the network by many sources, etc.
c. Finite packet error rate (duplicate or lost packets)—A result of the acknowledgment system in any store-and-forward discipline (this is a different use of the term "error rate" than in traditional telephony). Duplicate packets are caused when a node goes down after receiving a packet and forwarding it without having sent the acknowledgment. The previous node then generates a duplicate with its retransmission of the packet. Packets are lost when a node goes down after receiving a packet and acknowledging it before the successful transmission of the packet to the next node. An attempt to prevent lost and duplicate packets must fail as there is a tradeoff between minimizing duplicate packets and minimizing lost packets. If the nodes avoid duplication of packets whenever possible, more packets are lost. Conversely, if the nodes retransmit whenever packets may be lost, more packets are duplicated.
d. Disordering of packets—A property of the acknowledgment and routing algorithms.

These four properties describe what we term the *store-and-forward subnetwork.*

There are also two basic problems to be solved by the source and destination* in the virtual path described above:

e. Finite storage—A property of the nodes.
f. Differing source and destination bandwidths—Largely a property of the Hosts.

A slightly different treatment of this subject can be found in Reference 9.

The fundamental requirements for packet-switching networks are dictated by the six properties enumerated above. These requirements include:

a. Buffering—Buffering is required because it is generally necessary to send multiple data units on a communications path before receiving an acknowledgment. Because of the *finite delay* of the network, it may be desirable to have buffering for multiple packets in flight between source and destination in order to increase throughput. That is, a system without adequate buffering may have unacceptably low throughput due to long delays waiting for acknowledgment between transmissions.

b. Pipelining—The *finite bandwidth* of the network may necessitate the pipelining of each message flowing through the network by breaking it up into packets in order to decrease delay. The bandwidth of the circuits may be low enough so that forwarding the entire message at each node in the path results in excessive delay. By breaking the message into packets, the nodes are able to forward the first packet of the message through the network ahead of the later ones. For a message of P packets and a path of H hops, the delay is proportional to $P+H-1$ instead of $P * H$, where the proportionality constant is the packet length divided by the transmission rate.**

c. Error Control—The node-to-node packet processing algorithm must exercise error control, with an acknowledgment system in order to deal with the *finite packet error rate* of the circuits. It must also detect when a circuit becomes unusable, and when to begin to use it again. In the source-to-destination message processing algorithm, the destination may need to exercise some controls to detect missing and duplicated messages or portions of messages, which would appear as incorrect data to the end user. Further, acknowledgments of message delivery or non-delivery may be useful, possibly to trigger retransmission. This mechanism in turn requires error control and retransmission itself, since the delivery reports can be lost

* The question of whether the source and destination nodes or the source and destination Hosts should solve these problems is addressed in a later section.

** See page 90 of Reference 9 for a derivation and more exact result.

or duplicated. The usual technique is to assign some unique number to identify each data unit and to time out unanswered units. The error correction mechanism is invoked infrequently, as it is needed only to recover from node or line failures.

d. Sequencing—Since *packet sequences can be received out of order,* the destination must use a sequence number technique of some form to deliver messages in correct order, and packets in order within messages, despite any scrambling effect that may take place while several messages are in transit. The sequencing mechanism is frequently invoked since it is needed to recover from line errors.

e. Storage allocation—The fact that *storage* in the nodes is *finite* means that both the packet processing and message processing algorithms must exercise control over its use. The storage may be allocated at either the sender or the receiver.

f. Flow Control—The *different source and destination data rates* may necessitate implicit or explicit flow control rules to prevent the network from becoming congested when the destination is slower than the source. These rules can be tied to the sequencing mechanism, with no more messages (packets) accepted after a certain number, or tied to the storage allocation technique, with no more messages (packets) accepted until a certain amount of storage is free, or the rules can be independent of these features.

In satisfying the above six requirements, the algorithm often exercises contention resolution rules to allocate resources among several users. The twin problems of any such facility are:

- fairness—resources should be used by all users fairly;
- deadlock prevention—resources must be allocated so as to avoid deadlocks.

We have also come to believe that it is essential to have a reset mechanism to unlock "impossible" deadlocks and other conditions that may result from hardware or software failures.

Network performance goals

Packet-switching communications systems have two fundamental goals in the processing of data—low delay and high throughput. Each message should be handled with a minimum of waiting time, and the total flow of data should be as large as possible. The difference between low delay and high throughput is important. What the network user wants is the completion of his data transmission in the shortest possible time. The time between transmission of the first bit and delivery of the first bit is a function of network delay, while the time between delivery of the first bit and delivery of the last bit is a function of network throughput. For interactive users with short messages, low delay is more important, since there are few bits per message. For the transfer of long data files, high throughput is more important.

There is a fundamental tradeoff between low delay and high throughput, as is readily apparent in considering some of the mechanisms used to accomplish each goal. For low delay, a small packet size is necessary to cut transmission time, to improve the pipelining characteristics, and to shorten queueing latency at each node; furthermore, short queues are desirable. For high throughput, a large packet size is necessary to decrease the circuit overhead in bits per second and the processing overhead per bit. That is, long packets increase the effective circuit bandwidth and nodal processing bandwidth. Also, long queues may be necessary to provide sufficient buffering for full circuit utilization. Therefore, the network may need to employ separate mechanisms if it is to provide low delay for some users and high throughput for others.

To these two goals one must add two other equally important goals, which apply to message processing and to the operation of the network as a whole. First, the network should be cost-effective. Individual message service should have a reasonable cost as measured in terms of utilization of network resources; further, the network facilities, primarily the node computers and the circuits, should be utilized in a cost-effective way. Secondly, the network should be reliable. Messages accepted by the network should be delivered to the destination with a high probability of success. And the network as a whole should be a robust computer communications service, fault-tolerant, and able to function in the face of node or circuit failures.

In summary, we believe that delay, throughput, reliability, and cost are the four criteria upon which packet-switching network designs should be evaluated and compared. Further, it is the combined performance in all four areas which counts. For instance, poor delay and throughput characteristics may be too big a price to pay for "perfect" reliability.

Key design choices

We believe there are three major areas in which the key choices must be made in designing a packet-switching network. First, there is network hardware design, including the node computer, the network circuits, the Host-to-node connections, and overall connectivity. Second, there is store-and-forward subnetwork software design, primarily the routing algorithm and the node-to-node transmission procedures. Third, there is source-to-destination software design, which encompasses end-to-end transmission procedures and the division of responsibility between Hosts and nodes.* These topics are covered in the following sections.

* There are strong interactions between the topics discussed in the second and third areas. The end-to-end traffic requirements of a specific user can only be met if the store-and-forward subnetwork has mechanisms which act in concert with the source-to-destination mechanisms to provide the required performance. Discussion of this interaction, an important consideration in packet-switching network design, is beyond the scope of this paper.

NETWORK HARDWARE DESIGN

In this section we outline some of the design issues associated with the choice of the node computer, the network circuits, the Host-to-node connections, and overall connectivity. Since the factors affecting these choices change rapidly with the introduction of new technology, we discuss only general observations and design questions.

The node computer

The architecture of the node computer is related to several other network design parameters, as detailed below.

Processor

The speed of the processor is important in determining the throughput rates possible in the network. The store-and-forward processing bandwidth of the processor can be computed by counting instructions in the inner loop (see Reference 10 for an example). The source-to-destination processing bandwidth can be calculated in a similar fashion. These rates should be high enough so that the entire bandwidth of the network lines can be used, i.e., so that the node is not a bottleneck. It has been our experience that the speed of the processor and memory is the main factor in this bandwidth calculation; complex or specialized instruction sets are not valuable because simple instructions make up most of the node program.

A different aspect of the node computer which can also affect throughput is its responsiveness. Because circuits are synchronous devices, they require service with very tight time requirements. If the node does not notice that input has completed on a given circuit, and does not prepare for a new input within a given time, the next input arriving on that circuit will be lost. Likewise on output, the node must be responsive in order to keep the circuits fully loaded. This requirement suggests that some form of interrupt system[1] or high-speed polling device[11] is necessary to keep response latency low, and that the overhead of an operating system and task scheduler and dispatcher may be prohibitive. Finally, we note that the amount of time required by the node to process input and output is most critical in determining the *minimum* packet size, since it is with packets of this size that the highest packet arrival and departure rates (and thus processing requirements) can be observed. Of course, data buffering in the device interfaces can partially alleviate these problems.

Memory

The speed of memory may be a major determinant of processor speed, thus affecting the node bandwidth. An equally important consideration is memory speed for I/O transfers, since the node's overall bandwidth results from a division of total memory bandwidth based on some processing time for a given amount of I/O time. First, there is the question of whether the I/O transfers act in a cycle-stealing fashion to slow the processor or whether memory is effectively multi-ported to allow concurrent use. Then there is the issue of contention for memory among the various synchronous I/O devices. In a worst-case scenario, it is possible for all the I/O devices to request a memory transfer at the same instant, which keeps memory continuously busy for some time interval. A key design parameter is the ratio of this time to the available data buffering time of the least tolerant I/O device. This ratio should be less than one, and may therefore determine how much I/O can be connected to the node.

The size of the memory, naturally, is another key parameter. It has been our experience[1,10] that the program and associated data structures take up the majority of storage in the node. The remainder of memory is devoted to buffering of two kinds: packet buffering between adjacent nodes, and message buffering between source and destination nodes. These requirements can be calculated quite simply in each case as the product of the maximum data rate to be supported times the round trip time (for a returning acknowledgment). In large networks it may be necessary to rely on sophisticated compression techniques to ensure that tables for the routing algorithm, the source-to-destination transmission procedures, and so on, do not require excessive storage.

I/O

The speed of the I/O system has been touched upon above in relation to processor and memory bandwidth. Other factors worth noting are the internal constraints imposed by the I/O system itself—its delay and bandwidth. A different dimension, and one that we have found to be inadequately designed by most manufacturers, is the flexibility and extensibility of the I/O system. Most manufacturers supply only a limited range of I/O options (some of which may be too slow or too expensive to use). Further, only a limited number of each type can be connected. A packet-switching network node requires high performance from the I/O system, both in the number of connections and in their data rates.

General architecture

There are other factors to consider in evaluating or designing a node processor apart from performance in terms of bandwidth and delay. As we mentioned, extensibility in I/O is very important and comparatively rare; it is more common to find memory systems which can be expanded. Processor systems which can be expanded are not at all common, and yet processor bandwidth may be the limiting factor in some node configurations. Without a modular approach allowing processing, memory and I/O growth, the cost of the node computer can be quite high due to large step functions in component cost.

Another aspect of node computer architecture is its reliability, particularly for large systems with many lines and Hosts. A failure of such a system has a large impact on network performance. We have studied these issues of performance, cost, and reliability of node computers in a packet-switching network, and have developed, under ARPA sponsorship, a new approach to this problem. Our computer, called the Pluribus, is a multiprocessor made up of minicomputers with modular processors, memory, and I/O components, and a distributed bus architecture to interconnect them.[11] Because of its specially designed hardware and software features,[12] it promises to be a highly reliable system. We point out that many of these issues of performance, cost, and reliability could become critically important in very large networks serving thousands of Hosts and terminals.

We also note that there are so many stringent technical constraints on the computer that a choice made on other grounds (e.g., expediency, politics), as is common, is particularly unfortunate.

The network circuits

We next consider some of the important characteristics of the circuits used in the network.

Bandwidth

The bandwidth of the network circuits is likely to be their most important characteristic. It defines the traffic-carrying capacity of the network, both in the aggregate and between any given source and destination. What is less obvious is that the bandwidth (and hence the time to clock a packet out onto the line) may be the main factor determining the transit delays in the network. The minimum delay through the network depends mainly on circuit rates and lengths, and additional delays are largely accounted for by queueing delay, which is directly proportional to circuit bandwidth. These two factors lead to the general observation that the faster the network lines, the longer the packet should be, since long packets have less overhead and permit higher throughput, while the added delay due to length is less important at high circuits rates. In addition, more packet and message buffering is required when higher speed circuits are used.

Delay

The major effect of circuits with appreciable delay is that they require more buffering in the nodes to keep them fully loaded. That is, the node must maintain more packets in flight at once over a circuit with longer delay. This effect may be so large (a circuit using a satellite has a delay of a quarter of a second) as to require significantly more memory in the nodes.[10] This memory is needed at the nodes connected to the circuit to permit sufficient packet buffering for node-to-node transmission using the circuit. The subtle point is that additional buffering is also required at all nodes in the network that may need to maintain high source-to-destination rates over network paths which include this circuit. If they are to provide maximum throughput, they need sufficient message buffering to keep the entire network path fully loaded.

Reliability

Traditionally, the telephone carriers have quoted error rates in the following manner: "No more than an average of 1 bit in 10^6 bits in error." This definition is not entirely adequate for packet switching, though it may be for continuous transmission. For packet switching, the average bit error rate is less interesting than the average packet error rate (packets with one or more bits in error). For example, ten bits in error in every tenth packet is a 10 percent packet error rate, while one bit in error in every packet is a 100 percent packet error rate, yet the two cases have the same bit error rate.

An example of an acceptable statement of error performance would be as follows:

The circuit operates in two modes. Mode 1: no continuous sequence of packet errors longer than two seconds, with the average packet error rate less than one in a thousand. Mode 2: a continuous sequence of errors longer than two seconds with the following frequency distribution:

> 2 seconds	no more often than once per day
> 1 minute	no more often than once per week
>15 minutes	no more often than once per month
> 1 hour	no more often than once per 3 months
> 6 hours	no more often than once per year
> 1 day	never

While the figures above may seem too stringent in practice, the mode 1 bit error rate is actually quite lax compared to conventional standards. In any case, these are the kinds of behavior descriptions needed for intelligent design of packet-switching network error control procedures. Therefore, it is important that the carriers begin to provide such descriptions.

The packet error rate of a circuit has two main effects. First, if the rate is high enough, it can degrade the effective circuit bandwidth by forcing the retransmission of many packets. While this is basically a problem for the carrier to repair, the network nodes must recognize this condition and decide whether or not to continue to use the circuit. This is a tradeoff between reduced throughput with the circuit and increased delay and less network connectivity without it. Before the circuit can be used, it must be working in both directions for packets and for control information like routing and acknowledgments, and with a sufficiently low packet error rate.

The second effect of the error rate is present even for relatively low error rates. It is necessary to build a very good error-detection system so that the users of the network do not see errors more often than some specified extremely low frequency. That is, the network should detect enough errors so that the effective network error rate is at least an order of magnitude less than the Host error or failure rate. A usual technique here is a cyclic redundancy check on each packet. This checksum should be chosen carefully; to first order, its size does not depend on packet length* and it should be quite large, for example 24 bits for 50-Kbs lines and 32 bits for multi-megabit lines or lines with high error rates.

The Host-to-node connections

We examine the bandwidth and reliability of the Host connection to the network in the next two sections.

Bandwidth

The issues in choosing the bandwidth of the Host connections are similar to those for the network circuits. In addition to establishing an upper bound on the Hosts' throughput, the rate is also an important factor in delay. The delay to send or receive a long message over a relatively slow Host connection may be comparable in magnitude to the network round trip time. To eliminate this problem, and also to allow high peak throughput rates, the Host connection bandwidth should be as high as possible (within the limits of cost-effectiveness), even higher than the average Host throughput would indicate. By the same argument given above for packet size, a higher speed Host connection allows the use of a longer message with less overhead and Host processing per bit and therefore greater efficiency.

Reliability

The reliability of the Host connection is an important aspect of the network design; several points are worth noting. First, the connection should have a packet error rate which is at least as low as the network circuits. This can be accomplished by a highly reliable direct connection locally or by error-detection and retransmission. The use of error control procedures implies that the Host-node transmission procedures resemble the node-node transmission procedures which are discussed in a later section. Second, if the Host application requires extremely high reliability, a Host-to-Host data checksum and message sequence check are both useful for detecting infrequent network failures. Third, if the Host requires uninterrupted network service, and the Host is reliable enough itself to justify such service, multiple connections of the Host to various nodes can improve the availability of the network. This option complicates matters for the source-to-destination transmission procedures in the nodes (e.g., sequencing) since there may be more than one possible destination node serving the Host.

* Assuming that the probability of packet error is proportional to the product of packet length and bit error rate, the checksum length should be proportional to the log of the product of the desired time between undetected errors, the bit error rate, and the total bandwidth of all network circuits.

Overall connectivity

The subject of network topology is a complex one,[13] and we limit ourselves here to a few general observations. In practice, it seems that the connectivity of the nodes in the network should be relatively uniform. It is obvious that nodes with only a single line are to be avoided for reliability considerations but nodes with many circuits also present a reliability problem since they remove so much network connectivity when they are down. We also feel that the direction for future evolution of network geometries will be toward a "central office" kind of layout with relatively fewer nodes and with a high fan-in of nearby Hosts and terminals. This tendency will become more pronounced as higher reliability in the node computer becomes possible, even for large systems. One reason that we favor this approach is that a large node computer presents an increased opportunity for shared use of the node resources (processor and memory) among many different devices leading to a much more efficient and cost-effective implementation. This trend will mean that in the future, even more than now, a key cost of network topology will be the ultimate data connection to the user (Host or terminal), who may be far from the central office. Concentrators and multiplexors have been the traditional solution; in packet-switching networks, a small node computer could fill this function. In conclusion, we see flexibility and extensibility as two key requirements for the node computer. These factors together with increasing performance and fan-in requirements imply a very high reliability standard as well.

STORE-AND-FORWARD SUBNETWORK SOFTWARE DESIGN

We cover two major areas in our discussion of store-and-forward subnetwork software design, the routing algorithm and the node-to-node transmission procedures, both of which are packet-oriented and require no information about messages.

The routing algorithm

The fundamental step in designing a routing algorithm is the choice of the control regime to be used in the operation of the algorithm. *Non-adaptive* algorithms make no real attempt to adjust to changing network conditions; no routing information is exchanged by the nodes, and no observations or measurements are made at individual nodes. *Centralized adaptive* algorithms utilize a central authority which dictates the routing decisions to the individual nodes in response to network changes. *Isolated adaptive* algorithms operate independently with each node making exclusive use of local data to adapt to changing conditions. *Distributed adaptive* algorithms utilize internode cooperation and the exchange of information to arrive at routing decisions.*

Non-adaptive algorithms

Under this heading come such techniques as fixed routing, fixed alternate routing, and random routing (also known as flooding or selective flooding).

Simple fixed routing is too unreliable to be considered in practice for networks of more than trivial size and complexity. Any time a single line or node fails, some nodes become unable to communicate with other nodes. In fact, networks utilizing fixed routing always assume manual updates (as necessary) to another fixed routing pattern. However, in practice this means that every routine network component failure becomes a catastrophe for operational personnel, every site spending frantic hours manually reconstructing routing tables.[15]

At their best, in the absence of network component failure, fixed routing algorithms are inefficient. While the routing tables can be fixed to be optimal for some traffic flow, fixed routing is inevitably inefficient to the extent that network traffic flows vary from the optimal traffic flow. Unreliability and inefficiency are also characteristic of two alternative techniques to fixed routing which fall under the heading of non-adaptive algorithms: fixed routing with fixed alternate routes and random routing.[14]

Non-adaptive algorithms are all extremely simple and can therefore be implemented at low cost. They are thus possibly suitable for hardware implementation, for theoretical analysis and for studying the effects of varying other network parameters and algorithms.

In conclusion, we do not recommend non-adaptive routing for most networks because it is unreliable and inefficient. Despite these drawbacks, many networks have been proposed or begun with non-adaptive routing, generally because it is simpler to implement and to understand. Perhaps this tendency will be reversed as more information about other routing techniques is published and as network technology generally grows more sophisticated.

Centralized adaptive algorithms

In a centralized adaptive algorithm, the nodes send the information needed to make a routing decision to a Routing Control Center (RCC) which dictates its decision back to the nodes for actual use. The advantages claimed for a centralized algorithm are: (a) the routing computation is simpler to understand than a non-centralized algorithm, and the computation itself can follow one of several well known algorithms, e.g.;[16] (b) the nodes are relieved of the burden and overhead of the routing computation; (c) more nearly optimal routing is possible because of the sophistication that is possible in a centralized algorithm; and (d) routing "loops" (a possible temporary property of distributed algorithms) can be avoided.

Unfortunately, the processor bandwidth utilization at the center is likely to be very heavy. The classical algorithms that a centralized approach might use generally run in time proportional to N^3 (where N is the number of nodes in the network), while their distributed counterparts can run (through parallel execution) in time proportional to N^2. While it may be a saving to remove computation from the nodes, it may not be possible to perform a cubic computation on a large network in real time on a single computer, no matter how powerful.[14]

The claim that more optimal routing is possible with a centralized approach is not true in practice. To have optimal routing, the input information must be completely accurate and up-to-date. Of course, with any realistic centralized algorithm, the input data will no longer be completely accurate when it arrives at the center. Similarly, the output data—the routing decisions—will not go into effect at the nodes until some time after they have been determined at the center.

Distributed routing algorithms, whether fixed random, fixed alternate, or adaptive, may contain temporary loops, that is, a packet may traverse a complete circle while the algorithm adapts (or simulates adaptation in fixed strategies) to network change. Proponents of centralized routing often argue that such loops can best be avoided by centralization of the computation. However, because of the time lags cited above, there may indeed be loops during the time of propagation of a routing update when some nodes have adopted the new routes and other nodes have not.

A centralized routing algorithm has several inherent weaknesses in the updating procedure, the first being unreliability. If the RCC should fail, or the node to which it is connected goes down, or the lines around that node fail, or a set of lines and nodes in the network fail so as to partition the network into isolated components, then some or all of the nodes in the network are without any routing information. Of course, several steps can be taken to improve on the simple centralized policy. First, the RCC can have a backup computer, either doing another task until a RCC failure, or else on hot standby. This is not sufficient to meet the problem of network failures, only local outages, but it is necessary if the RCC computer has any appreciable failure rate. Second, there can be multiple RCCs in different locations throughout the network, and again the extra computers can be in passive or active standby. Here there is the problem of identifying which center is in control of which nodes, since the nodes must know to which center to send their routing input data.

* A much more detailed discussion is given in Reference 14.

A related difficulty with centralized algorithms lies in the fact that when a node or line fails in the network, the failed component may have been on the previously best path between the RCC and the nodes trying to report the failure. In this case, just at the time the RCC needs routes over which to receive and transmit routing information, no routes are available; the availability of new routes requires the very change the RCC is unsuccessfully attempting to distribute. Solutions which have been proposed to solve this "deadlock" are slow, complicated, awkward, and frequently rely on the temporary use of distributed algorithms.[17]

Finally, centralized algorithms can place heavy and uneven demands on network line bandwidth; near the RCC there is a concentration of routing information going to and from the RCC. This heavy line utilization near the center means that centralized algorithms do not grow gracefully with the size of the network and, indeed, this may place an upper limit on the size of the network.

Isolated adaptive algorithms

One of the primary characteristics of an isolated algorithm which attempts to adapt to changing conditions is that it takes on the character of a heuristic process: it must "learn" and "forget" various facts about the network environment. While such an approach may have an intuitive appeal, it can be shown rather simply that heuristic routing procedures are unstable and are therefore not of interest for most practical network applications. The fundamental problem with isolated adaptive algorithms is that they must rely on indirect information about network conditions, since each node operates independently and without direct knowledge of or communication with the other nodes.

There are two basic approaches to be employed, separately or in tandem, to the process of learning and forgetting. We call these approaches positive feedback and negative feedback. One way to implement positive feedback was suggested by Baran as part of his hot-potato routing doctrine.[18] Each node increments the handover number in a packet as it forwards the packet. Then the handover number is used in a "backwards learning" technique to estimate the transit time from the current node to the source of the packet. Clearly, this scheme has drawbacks because it lacks any *direct* way of adapting to changes. If no packets from a given source are routed through a node by the rest of the network, the node has no information about which route to choose in sending a message to that source. In general, as part of a positive feedback loop, the routing algorithm must periodically try routes other than the current best ones, since it has no direct way of knowing if better routes exist. Thus, there must always be some level of traffic traveling on any route that the nodes are to learn about, since it is only by feedback from traffic that they can learn.

The other half of an adaptive isolated algorithm is the negative feedback cycle. One technique to use here is to penalize the choice of a given path when a packet is detected to have returned over the same path without being delivered to its destination. The relation of this technique to the exploratory nature of positive feedback is evident.

An adaptive isolated algorithm, therefore, has this fundamental weakness: in the attempt to adapt heuristically, it must oscillate, trying first one path and then another, even under stable network conditions. This oscillation violates one of the important goals of any routing algorithm, stability, and it leads to poor utilization of network resources and slow response to changing conditions. Incorrect routing of the packets during oscillation increases delay and reduces effective throughput correspondingly. There is no solution to the problem of oscillation in such algorithms. If the oscillation is damped to be slow, then the routing will not adapt quickly to improvements and will therefore declare nodes unreachable when they are not, with the result that suboptimal paths will be used for extended periods. If the oscillation is fast, then suboptimal paths will also be used much of the time, since the network will be chronically full of traffic going the wrong way.

Distributed adaptive algorithms

In our experience, distributed adaptive algorithms have none of the inherent limitations of the above algorithms; e.g., not the inherent unreliability and inefficiency of nonadaptive algorithms, nor the unreliability and size limitations of centralized algorithms, nor the inherent inefficiency and instability of isolated algorithms. For example, the distributed adaptive routing algorithm in the ARPA Network has operated for five years with little difficulty and good performance. However, distributed algorithms do have some practical difficulties which must be overcome in order to obtain good performance.

Consider the following example of a distributed adaptive algorithm. Each node estimates the "distance" it expects a packet to have to traverse to reach each possible destination over each of its output lines. Periodically, it selects the minimum distance estimate for each destination and passes these estimates to its immediate neighbors. Each node then constructs its own routing table by combining its neighbors' estimates with its own estimates of distance to each neighbor. For each destination, the table is then made to specify that selected output line for which the sum of the estimated distance to the neighbor plus the neighbor's distance estimate to the destination is smallest.

Such an algorithm can be made to measure distance in hops (i.e., lines which must be traversed), delay, or any of a number of other metrics including excess bandwidth and reliability (of course, for the latter two, one must maximize rather than minimize). The above algorithm is representative of a class of distributed adaptive algorithms which we consider briefly in the remainder of this section. For simplicity of discussion we will assume that distance is measured in hops.

The first point is that distributed algorithms are slow in adapting to some kinds of change; in particular, the algorithm reacts quickly to good news, and slowly to bad news. If the number of hops to a given node decreases, the nodes soon all agree on the new, lower, number. If the hop count increases, the nodes will not believe the reports of higher counts while they still have neighbors with the old, lower values. This is demonstrated in Reference 14. Another point is that there is no way for a node to know ahead of time what the next-best or fall-back path will be in the event of a failure, or indeed if one exists. In fact, there must be some finite time, the network response time, between when a change in the network occurs and when the routing algorithm adapts to the change. This time depends on the size and shape of the network.

We have come to conclude that the routing algorithm should continue to use the best route to a given destination, both for updating and forwarding, for some time period after it gets worse. That is, the algorithm should report to the adjacent nodes the current value of the previous best route and use it for routing packets for a given time interval. We call this *hold down.*[14] One way to look at this is to distinguish between changes in the network topology and traffic that necessitate changing the choice of the best route, and those changes which merely affect the characteristics of the route, like hop count, delay, and throughput. In the case when the identity of the path remains the same, the mechanism of hold down provides an instantaneous adaptation to the changes in the characteristics of the path; certainly, this is optimal. When the identity of the path must change, the time to adapt is equal to the absolute minimum of one network response time, while the other nodes have a chance to react to the worsening of the best path and to decide on the next best path. This is optimal for any algorithm within the practical limits of propagation times.*

The routing algorithm is extremely important to network reliability, since if it malfunctions the network is useless. Further, a distributed routing algorithm has the property that all the nodes must be performing the routing computation correctly for the algorithm to be reliable. A local failure can have global consequences; e.g., one node announcing that it is the best path to all nodes. Routing messages between nodes must have checksums and must be discarded if a checksum error is detected. All routing *programs* must be checksummed before every execution to verify that the code about to be run is correct. The checksum of the program should include the preliminary checksum computation itself, the routing program, any constants referenced, and anything else which could affect its successful execution. Any time a checksum error is detected in a node, the node should immediately be stopped from participating in the routing computation until it is restored to correct operation again.

* This is a very simplified description of hold down. A more complete description states in detail when hold down should be invoked and for what duration. Such a description may be found in Reference 14 and more is being learned.[19]

Node-to-node transmission procedures

In this section we discuss some of the issues in designing node-to-node transmission procedures, that is, the packet processing algorithms. We touch on these points only briefly since many of them are simple or have been discussed previously. Note that many of these issues occur again in the discussion of source-to-destination transmission procedures.

Buffering and pipelining

As we noted in discussing memory requirements, the amount of node-to-node packet buffering needs to equal the product of the circuit rate times the expected acknowledgment delay in order to get full line utilization. It may also be efficient to provide a small amount of additional buffering to deal with statistical fluctuations in the arrival rates, i.e., to provide queueing. These requirements imply that the nodes must do bookkeeping about multiple packets, which raises the several issues discussed next.

Error control

We have discussed many of the aspects of node-to-node error control above: the need for a packet checksum, its size, the basis of the acknowledgment/retransmission system, the decision on whether the line is usable, and so on. These procedures are critical for network reliability, and they should therefore run smoothly in the face of any kind of node or circuit failure. Where possible, the procedures should be self-synchronizing; at least they should be free from deadlock and easy to resynchronize.[10]

Storage allocation and flow control

Storage allocation can be fairly simple for the packet processing algorithms. The sender must hold a copy of the packet until it receives an acknowledgment; the receiver can accept the packet if it is without error and there is an available buffer. The receiver should not use the last free buffer in memory, since that would cut off the flow of control information such as routing and acknowledgments. In accepting too many packets, there is also the chance of a storage-based deadlock in which two nodes are trying to send to each other and have no more room to accept packets. This is explained fully in Reference 20.

The above implies that the flow control procedures can also be fairly simple. The need to buffer a circuit can be expressed in a quantitative limit of a certain number of packets. Therefore, the node can apply a cut-off test per line as its flow control throttle. More stringent rules can be used, but may be unnecessary.

Priority

The issue of priority in packet processing is quite important for network performance. First of all, the concept of two or more priority levels for packets is useful in decreasing queueing delay for important traffic. Beyond this, however, careful attention must be paid to other kinds of transmissions. Routing messages should go with the highest priority, followed by acknowledgments (which can also be piggybacked in packets). Packet retransmissions must be sent with the next highest priority, higher than that for first transmission of packets. If this priority is not observed, retransmissions can be locked out indefinitely. The question of preemptive priority (i.e., stopping a packet in mid-transmission to start a higher priority one) is one of a direct tradeoff of bandwidth against delay since circuit bandwidth is wasted by each preemption.

Packet size

There has been much thought given in the packet-switching community to the proper size for packets. Large packets have a lower probability of successful transmission over an error-prone telephone line (and this drives the packet size down), while overhead considerations (longer packets have a lower percentage overhead) drive packet size up. The delay-lowering effects of pipelining become more pronounced as packet size decreases, generally improving store-and-forward delay characteristics; further, decreasing packet size reduces the delay that priority packets see because they are waiting behind full length packets. However, as the packet size goes down, effective throughput also goes down due to overhead. Metcalfe has previously commented on some of these points.[21]

Kleinrock and Naylor[22] recently suggested that the ARPA Network packet size was suboptimal and should perhaps be reduced from about 1000 bits to 250 bits. This was based on optimization of node buffer utilization for the observed traffic mix in the network. However, in Reference 23, we point out that the relative cost of node buffer storage vs. circuits is possibly such that one should not try to optimize node buffer storage. The true trade-off which governs packet size might well be efficient use of phone line bandwidth (driving packet size larger) vs. delay characteristics (driving packet size smaller). If buffer storage is limiting, perhaps one should just buy more. Further, it is probably true that if one is trying for high bandwidth utilization, buffer size must be large. That is, high bandwidth utilization probably implies the use of large packets, which implies full buffers; when idle, the buffer size does not matter.

As noted above, the choice of packet is influenced by many factors. Since some of the factors are inherently in conflict, an optimum is difficult to define. much less find. The current ARPA Network packet size of about 1000 bits is a good compromise. Other packet sizes (e.g., the 2000 bits used in several other networks) may also be acceptable compromises. However. note that a 2000-bit packet size generally means a factor of two increase in delay over a 1000-bit packet size, because even high priority short packets will be delayed behind normal long packets which are in transmission at each node. The use of preemptive priority might make longer packet sizes efficient.

Davies and Barber[24] are often quoted as recommending a minimum length "packet" of about 2000 bits because they have concluded that most of the messages currently exchanged within banks and airlines fit nicely in one packet of this size. To clarify this point, we note that they use the term "packet" for the unit of information we call a "message" and thus are not actually addressing the issue of packet size. We discuss message size below.

SOURCE-TO-DESTINATION SOFTWARE DESIGN

In this section we discuss the end-to-end transmission procedures and the division of responsibility between the Hosts and nodes.

End-to-end transmission procedures

There is a considerable controversy at the present time over whether or not a store-and-forward subnetwork of nodes should concern itself with end-to-end transmission procedures. Many workers[2] feel that the subnetwork should be close to a pure packet carrier with little concern for maintaining message order, for high levels of correct message delivery, for message buffering in the subnetwork, etc. Other workers, including ourselves,[23] feel that the subnetwork should take responsibility for many of the end-to-end message processing procedures. Of course, there are some workers who hold to positions in between.[3] However, many design issues remain constant whether these functions are performed at Host level or subnetwork level, and we discuss these constants in this section.

Buffering and pipelining

As noted earlier in this paper, any practical network must allow multiple messages simultaneously in transit between the source and the destination, to achieve high throughput. If, for example, one message of 2000 bits is allowed to be outstanding between the source and destination at a time, and the normal network transit for the message including destination-to-source acknowledgment is 100 milliseconds, then the throughput rate that can be sustained is 20,000 bits per second. If slow lines, slow responsiveness of the destination Host, great distance, etc., cause the normal network transit time to be half a second, then the throughput rate is reduced to only 4,000 bits per second. Likewise, we think that pipelining is essential for most networks to improve delay characteristics; data should travel in reasonably short packets.

To summarize, low delay requirements drive packet size smaller, network and Host lines faster, and network paths shorter (i.e., fewer node-to-node hops). High throughput requirements drive the number of packets in flight up, packet overhead down, and the number of alternative paths up.

Error control

We consider source-to-destination error control to comprise three tasks: detecting bit errors in the delivered messages, detecting missing messages or pieces of messages, and detecting duplicate messages or pieces of messages.

The former task is done in a straightforward manner through the use of checksums. A checksum is appended to the message at the source and the checksum is checked at the destination; when the checksum does not check at the destination, the incorrect message is discarded, requiring it to be retransmitted from the source. Several points about the manner in which checksumming should be done are worthy of note: (a) If possible, the checksum should check the correctness of the resequencing of the messages which possibly got out of order in their traversal of the network. (b) A powerful checksum is more efficient than alternative methods such as replication of a critical control field; it is better to extend the checksum by the number of bits that would have been used in the redundant field. (c) Unless encryption is desirable for some other reason it is simpler (and just as safe) to prevent delivery of a message to an incorrect Host through the use of a powerful checksum than it is to use an encryption mechanism. (d) Node-to-node checksums do not fulfill the same function as end-to-end checksums because they check only the lines, not the nodes.

An inherent characteristic of packet-switching networks is that some messages or portions of messages (i.e., packets) will fail to be delivered, and there will be some duplicate delivery of messages or portions of messages, as described in the section on network properties.*

Missing messages can be detected at the destination through the use of one state bit for each unit of information which can be simultaneously traversing the network. An interesting detail is that for the purposes of missing message detection, the state bits used must precisely cycle through all possible states. For example, stamping messages with a time stamp does nothing for the process of missing message detection because, unless a message is sent for every "tick" of the time stamp, there is no way to distinguish the case of a missing message from the case where no messages were sent for a time.

Duplicate messages can be detected with an identifying sequence number such that messages which arrive from a prior point in the sequence are recognized as duplicates. What should be noted carefully here is that duplicate messages can arrive at the destination up to some time, possibly quite long, after the original copy, and the sequence number must not complete a full cycle during this period. For example, if a network goal is to be able to transmit 200 minimum length messages per second from the source to the destination and each needs a unique sequence number, and if it is possible for messages to arrive at the destination up to 15 seconds after initial transmission from the source, then the sequence number must be able to uniquely identify at least 3000 packets. It is usually no trouble to calculate the maximum number of messages that can be sent during some time interval. What is more difficult is to limit the maximum time after which duplicate messages will no longer arrive at the destination. One method is to put a timer in each message which is counted down as the message traverses the network; if the timer ever counts out, the message is discarded as too old, thus guaranteeing that no messages older than the initial setting of the timer will be delivered to the destination. Alternatively, one can calculate approximately the maximum arrival time through study of all the worst case paths through the network and all the worst case combinations of events which might cause messages to loop around in the network for excessive lengths of time; this seems to work reasonably well in practice.

In either case, there certainly must be mechanisms to resynchronize the sequence numbers between the source and the destination at node start-up time, to recover from a node failure, etc. A good practice is to resynchronize the sequence numbers occasionally even though they are not known to be out of step. A good frequency with which to do redundant resynchronization would be every time a message has not been sent for longer than the maximum delivery time. In fact, this is the maximum frequency with which the resynchronization can be done (without additional mechanisms); if duplicates are to be detected reliably, the sequence number at the destination *must* function without disruption for the maximum delivery time after the "last message" has been sent. If it is desirable or necessary to resynchronize the sequence numbers more often than the maximum time, an additional "use" number must be attached to the sequence number to uniquely identify which "instance" of this set of sequence numbers is in effect; and, of course, the packets must also carry the use number. This point is addressed in greater detail in References 25 and 26.

The next point to make about end-to-end error control is that any message going from source to destination can potentially be missing or duplicated; i.e., not only data messages but control messages. In fact, the very messages used in error control (e.g., sequence number resynchronization messages) can themselves be missing or duplicated, and a proper end-to-end protocol must handle these cases.

Finally, there must be some inquiry-response system from the source to the destination to complete the process of detecting lost messages. When the proper reply or acknowledgment has not been received for too long, the source may inquire whether the destination has received the message in question. Alternatively, the source may simply retransmit the message in question. In any case, this source inquiry and retransmission system must also function in the face of duplicated or lost inquiries and inquiry response control messages. As with the inter-node acknowledgment and retransmission system, the end-to-end acknowledgment and retransmission system must depend on positive acknowledgments from the destination to the

* Throughout the remainder of this subsection we use the word "message" to mean either messages or portions of messages (i.e., packets).

source and on explicit inquiries or retransmissions from the source. Negative acknowledgments from the destination to the source are never sufficient (because they might get lost) and are only useful for increased efficiency.

Storage allocation and flow control

One of the fundamental rules of communications systems is that the source cannot simply send data to the destination without some mechanism for guaranteeing storage for that data. In very primitive systems one can guarantee a rate of disposal of data, as to a line printer, and not exceed that rate at the data source. In more sophisticated systems there seem to be only two alternatives. Either one can explicitly reserve space at the destination for a known amount of data in advance of its transmission, or one can declare the transmitted copy of the data expendable, sending additional copies from the source until there is an acknowledgment from the destination. The first alternative is the high bandwidth solution: when there is no space, only tiny messages travel back and forth between the source and destination for the purpose of reserving destination storage. The second alternative is the low delay solution: the text of the message propagates as fast as possible. See Reference 10 for a more lengthy discussion.

In either case storage is tied up for an amount of time equal to at least the round trip time. This is a fundamental result—the minimum amount of buffering required by a communications system, either at the source or at the destination, equals the product of round trip time and the channel bandwidth. The only way to circumvent this result is to count on the destination behaving in some predictable fashion (an unrealistic assumption in the general case of autonomous communicating entities).

As we stated earlier, our experience and analysis convince us that if both low delay and high throughput are desired, then there must be mechanisms to handle each, since high throughput and low delay are conflicting goals. This is true, in particular, for the storage allocation mechanism. In several networks, e.g.,[7] mainly for the sake of simplicity, only the low delay solution has been proposed or implemented; that is, messages are transmitted from the source without reservation of space at the destination. Those people making the choice never to reserve space at the destination frequently assert that high bandwidth will still be possible through use of a mechanism whereby the source sends messages toward the destination, notes the arrival of acknowledgments from the destination, uses these acknowledgments to estimate the destination reception rate, and adjusts its transmissions to match that rate. We feel that such schemes may be quite difficult to parameterize for efficient control and therefore may result in reduced effective bandwidth and increased effective delay. If, in addition to possible discards at the destination, the network solves its internal problems by discarding packets, or if the destination Host too often solves its internal problems by discarding packets, performance will suffer further. As reported in Reference 20, contention for destination storage, which must be resolved through the discard of packets in the absence of a storage allocation mechanism, happens practically continuously under even modest traffic loads, and in a way uncoordinated with the rates and strategies of the various sources. As a result, well-behaved Hosts may unavoidably be penalized for the actions of poorly-behaved Hosts.

In addition to space to hold all data, there must also be space to hold all control messages. In particular, there must be space to record what needs to be sent and what has been sent. If a message will result in a response, there must be space to hold the response; and once a response has been sent, the information about what kind of answer was sent must be kept for as long as retransmission of that response may be necessary.

Precedence and preemption

The first point to note about precedence and preemption is that the total transit time being specified for most packet-switching networks of which we are aware is on the order of less than a few seconds (often only a fraction of a second). Thus, the traditional specifications (for example, low priority traffic must be able to preempt all other traffic so that it can traverse the network in under two minutes) no longer make much sense. When all messages traverse the network in less than a few seconds, there is generally no need to specify that top priority traffic must preempt other traffic, nor to specify the relative precedences between the other types of traffic.

Though priority is not strictly necessary for speed, it may be useful for contention resolution. It appears to us that there are three precedence and preemption strategies that are reasonable to consider for a packet-switching network. Strategy 1 is to permanently assign the resources necessary to handle high priority traffic; this guarantees the delivery time for the high priority traffic but is expensive and should only be done for limited high priority traffic. Strategy 2 is to preempt resources as necessary for high priority traffic. This can have two effects. Preempting packet buffers results in data loss; preempting internal node tables (e.g., the tables associated with packet sequence numbering) results in state information loss. State information loss means that data errors are possible which may go unreported. Strategy 3 is not to preempt resources, and to rely on the standard mechanisms with a priority ordering. This is simple for the nodes, but it does not itself guarantee delivery within a certain time.

We think the correct strategy is probably a mixture of the strategies above. Possibly some resources, on a very limited basis, should be reserved for the tiny amount of flash traffic. This guarantees minimum delay without any queueing latency. For the rest of the traffic, the normal delivery times are probably acceptable. The presence of higher priority traffic can cause gradual throttling of lower priority traffic, without loss of state information. As the time to do this graceful throttling is normally only a fracing the main Host from coping with these tasks. Stated reasons for this notion include:

- It is difficult to change the monitor in the main Host, and new monitor releases by the Host manufacturer pose continuing compatibility problems.
- Core or timing limitations exist in the main Host.
- It is desirable to use I/O arrangements that may already exist or be available between the main Host and the additional mini (and between the mini and the node) to avoid design or procurement of new I/O gear for the main Host.

While this approach may sound good in principle, and, in fact, may be the only possible approach in some instances, it often leads to problems.

First, the I/O arrangements between the main Host and any preexisting peripheral processor were not designed for network connection and usually present timing and bandwidth constraints that greatly degrade performance. More seriously, the logical protocols that may have preexisted will almost certainly preclude the main Host from acting as a general purpose Host on the network. For instance, while initial requirements may only indicate a need for simple file transfers to a single distant point, requirements tend to change in the face of new facilities, and the network cannot then be used to full advantage.[29]

Second, the peripheral processor and its software are often provided by an outside group, and the Host organization may know even less about their innards than they know about the main Host. The node is centrally maintained, improved, modified, and controlled by the Network Manager, but the peripheral processor, while an equally foreign body, is not so fortunate. This issue alone is crucial; functions that do not belong in the main Hosts belong in centrally monitored network equipment. Note that it is exactly those Host groups who are unwilling to touch the main Host's monitor who will be unlikely to be able to make subtle improvements in the protocols, error message handling, and timing of the peripheral processor. From a broader economic view, common functions belong in the network and should be designed once; the peripheral processor approach is a succession of costly special cases and the total cost is greatly escalated.

The long term solution to the dilemma is to have the various manufacturers support hardware and software interfaces that connect to widely used networks. This is not likely to occur until commercial networks exist and are widely available. In the meantime, potential Host organizations that wish to use early networks (like the ARPA Network) should try to find ways to put the network connection directly into the main Host. An anthropomorphic illustration may be helpful: the network is, among other things, a set of standardized protocols or languages. A potential network Host is in the position of a person who needs to have dealings with people who speak a language he does not know. If he does not want to learn the language, he can indeed opt for using an interpreter, but performance is poor, the process is very inconvenient, expensive, and unpleasant, and subtle meaning is always lost. The situation is quite similar when a Host tries to work through a peripheral processor. If a Host wishes to interact with a network, it is usually unrealistic to try to make the Host think that the network is a card reader or some other familiar peripheral. As usual, you get what you pay for.

Other message services

One commonly suggested design requirement is for storage in the communications subnetwork, usually for messages which are currently undeliverable because a Host or a line is down. This requirement should have no effect whatsoever on the design of the communications part of the network; it is an orthogonal requirement which should be implemented by providing special storage Hosts at strategic locations in the network. These can be at every node, at a few nodes, or at a single node, depending on the relative importance of reliability, efficient line utilization, and cost.

Another commonly suggested design requirement is for the communications subnetwork to provide a message broadcast capability; i.e., a Host gives a message to its node along with a list of Host addresses and the nodes somehow send copies to all the Hosts in the list. Again we believe that such a requirement should have no effect on the design of the communications part of the network and that messages to be broadcast should be sent to a special Host (perhaps one of the ones in the previous paragraph) for such broadcast.

CONCLUSION

There has now been considerable experience with the design of packet-switching networks and several groups (ours included) believe that they have come to understand many of the fundamental design issues. On the other hand, packet switching is still in its youth, and there are many new issues to be explored. Such new issues include, among others: (a) the techniques for transferring packet-switching technology from its initial limited R&D implementations to widespread production implementations; (b) the methods whereby the newly available packet-by-satellite technology can be utilized in packet-switching networks; (c) transmission of speech through packet-switching networks; (d) packet transmission by radio; (e) interconnection of packet-switching networks; and (f) effects of packet-switching networks on Host operating system design. Several other papers in these same proceedings cover in detail some of the new design issues just mentioned[30,31,32,33] and we plan to address some of these new issues ourselves in the near future.

ing the main Host from coping with these tasks. Stated reasons for this notion include:

- It is difficult to change the monitor in the main Host, and new monitor releases by the Host manufacturer pose continuing compatibility problems.
- Core or timing limitations exist in the main Host.
- It is desirable to use I/O arrangements that may already exist or be available between the main Host and the additional mini (and between the mini and the node) to avoid design or procurement of new I/O gear for the main Host.

While this approach may sound good in principle, and, in fact, may be the only possible approach in some instances, it often leads to problems.

First, the I/O arrangements between the main Host and any preexisting peripheral processor were not designed for network connection and usually present timing and bandwidth constraints that greatly degrade performance. More seriously, the logical protocols that may have preexisted will almost certainly preclude the main Host from acting as a general purpose Host on the network. For instance, while initial requirements may only indicate a need for simple file transfers to a single distant point, requirements tend to change in the face of new facilities, and the network cannot then be used to full advantage.[29]

Second, the peripheral processor and its software are often provided by an outside group, and the Host organization may know even less about their innards than they know about the main Host. The node is centrally maintained, improved, modified, and controlled by the Network Manager, but the peripheral processor, while an equally foreign body, is not so fortunate. This issue alone is crucial; functions that do not belong in the main Hosts belong in centrally monitored network equipment. Note that it is exactly those Host groups who are unwilling to touch the main Host's monitor who will be unlikely to be able to make subtle improvements in the protocols, error message handling, and timing of the peripheral processor. From a broader economic view, common functions belong in the network and should be designed once; the peripheral processor approach is a succession of costly special cases and the total cost is greatly escalated.

The long term solution to the dilemma is to have the various manufacturers support hardware and software interfaces that connect to widely used networks. This is not likely to occur until commercial networks exist and are widely available. In the meantime, potential Host organizations that wish to use early networks (like the ARPA Network) should try to find ways to put the network connection directly into the main Host. An anthropomorphic illustration may be helpful: the network is, among other things, a set of standardized protocols or languages. A potential network Host is in the position of a person who needs to have dealings with people who speak a language he does not know. If he does not want to learn the language, he can indeed opt for using an interpreter, but performance is poor, the process is very inconvenient, expensive, and unpleasant, and subtle meaning is always lost. The situation is quite similar when a Host tries to work through a peripheral processor. If a Host wishes to interact with a network, it is usually unrealistic to try to make the Host think that the network is a card reader or some other familiar peripheral. As usual, you get what you pay for.

Other message services

One commonly suggested design requirement is for storage in the communications subnetwork, usually for messages which are currently undeliverable because a Host or a line is down. This requirement should have no effect whatsoever on the design of the communications part of the network; it is an orthogonal requirement which should be implemented by providing special storage Hosts at strategic locations in the network. These can be at every node, at a few nodes, or at a single node, depending on the relative importance of reliability, efficient line utilization, and cost.

Another commonly suggested design requirement is for the communications subnetwork to provide a message broadcast capability; i.e., a Host gives a message to its node along with a list of Host addresses and the nodes somehow send copies to all the Hosts in the list. Again we believe that such a requirement should have no effect on the design of the communications part of the network and that messages to be broadcast should be sent to a special Host (perhaps one of the ones in the previous paragraph) for such broadcast.

CONCLUSION

There has now been considerable experience with the design of packet-switching networks and several groups (ours included) believe that they have come to understand many of the fundamental design issues. On the other hand, packet switching is still in its youth, and there are many new issues to be explored. Such new issues include, among others: (a) the techniques for transferring packet-switching technology from its initial limited R&D implementations to widespread production implementations; (b) the methods whereby the newly available packet-by-satellite technology can be utilized in packet-switching networks; (c) transmission of speech through packet-switching networks; (d) packet transmission by radio; (e) interconnection of packet-switching networks; and (f) effects of packet-switching networks on Host operating system design. Several other papers in these same proceedings cover in detail some of the new design issues just mentioned[30,31,32,33] and we plan to address some of these new issues ourselves in the near future.

ACKNOWLEDGMENTS

Since 1969 our research on packet-switching network design has been encouraged and supported by the Information Processing Techniques office of the Advanced Research Projects Agency. Many of our colleagues at Bolt Beranek and Newman Inc. have participated in our research. Through the ARPA Network and through the International Network Working Group we have been fortunate to be able to receive frequently incisive critiques of our work and to have the opportunity to study the work of others. In particular, over the past year our interactions with Holger Opderbeck and his colleagues at UCLA have been enlightening. Finally, we acknowledge the substantial assistance of Robert Brooks and Barbara Erwin with the preparation of the manuscript, and we acknowledge the very useful comments of Drs. J. Burchfiel, W. Hawyrlko, R. Metcalfe, and J. Postel, who reviewed the presentation and content of this paper.

REFERENCES

1. Heart, F. E., R. E. Kahn, S. M. Ornstein, W. R. Crowther, and D. C. Walden, "The Interface Message Processor for the ARPA Computer Network," *AFIPS Conference Proceedings*, Vol. 36, June 1970, pp. 551-567; also in *Advances in Computer Communications*, W. W. Chu (ed.), Artech House Inc., 1974, pp. 300-316.
2. Pouzin, L., "Presentation and Major Design Aspects of the Cyclades Computer Network," *Proceedings of the Third ACM Data Communications Symposium*, November 1973, pp. 80-88.
3. Brant, G. J. and G. J. Chretien, "Methods to Control and Operate a Message-Switching Network," *Computer-Communications Network and Teletraffic*, Polytechnic Press of the Polytechnical Institute of Brooklyn, Brooklyn, N.Y., 1972.
4. Barber, D. L. A. (ed.), *A Specification for a European Informatics Network*, Co-Opèration Europèenne dans le Domaine de la Recherche Scientifique et Technique, January 4, 1974.
5. Rosner, R. D., "A Digital Data Network Concept for the Defense Communications System," *Proceedings of the National Telecommunications Conference*, Atlanta, November 1973, pp. 22C1-6.
6. Davies, D. W., K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson, "A Digital Communication Network for Computers Giving Rapid Response at Remote Terminals," *Proceedings of the ACM Symposium on Operating Systems Principles*, October 1967.
7. Auerbach Publishers Inc., *Public Packet Switching Networks*, Data Processing Manual No. 3-08-04, 1974.
8. Despres, R. F., "A Packet Switching Network with Graceful Saturated Operation," *Proceedings of the First International Conference on Computer Communication*, October 1972, pp. 345-351.
9. Pouzin, L., *Basic Elements of a Network Data Link Control Procedure (NDLC)*. INWG 54. NIC 30375, January 1974, a limited number of copies available for the cost of reproduction and handling from INWG, c/o Prof. V. Cerf. Digital Systems Laboratory. Stanford, CA. 94305.
10. McQuillan, J. M., W. R. Crowther, B. P. Cosell, D. C. Walden, and F. E. Heart, "Improvements in the Design and Performance of the ARPA Network." *AFIPS Conference Proceedings*. Vol. 41, December 1972, pp. 741-754.
11. Heart, F. E., S. M. Ornstein. W. R. Crowther, and W. B. Barker, "A New Minicomputer Multiprocessor for the ARPA Network," *AFIPS Conference Proceedings*, Vol. 42, June 1973, pp. 592-537; also in *Selected Papers: International Advanced Study Institute, Computer Communication Networks*, R. L. Grimsdale and F. F. Kuo (eds.), University of Sussex, Brighton, England, September 1973; also in *Advances in Computer Communications*, W. W. Chu (ed.), Artech House Inc., 1974, pp. 329-337.
12. Ornstein, S. M., W. R. Crowther, M. F. Kraley, R. D. Bressler, A. Michel, and F. E. Heart, "Pluribus—A Reliable Multiprocessor," these proceedings.
13. Frank, H., R. E. Kahn, and L. Kleinrock, "Computer Communications Network Design—Experience with Theory and Practice," *AFIPS Conference Proceedings*, Vol. 40, June 1972, pp. 255-270; also in *Networks*, Vol. 2, No. 2, 1972, pp. 135-166; also in *Advances in Computer Communication*, W. W. Chu (ed.), Artech House Inc., 1974, pp. 254-269.
14. McQuillan, J. M., *Adaptive Routing Algorithms for Distributed Computer Networks*, BBN Report No. 2831, May 1974, available from the National Technical Information Service, AD781467.
15. Grange, J. L., *Cyclades Network*, personal communication.
16. Floyd, R. W., "Algorithm 97, Shortest Path," *CACM* 5 (6), June 1962, p. 345.
17. Gerla, M., "Deterministic and Adaptive Routing Policies in Packet-Switched Computer Networks, *Proceedings of the Third ACM Data Communications Symposium*, November 1973, pp. 23-28.
18. Baran, P., *On Distributed Communications: I. Introduction to Distributed Communications Networks*, Rand Corp. Memo RM-3420-PR, August 1964, p. 37.
19. Opderbeck, H. and W. Naylor, *ARPA Network Measurement Center*, personal communication.
20. Kahn, R. E. and W. R. Crowther, "Flow Control in a Resource-Sharing Computer Network," *Proceedings of the Second ACM/IEEE Symposium on Problems in the Optimization of Data Communications Systems*, Palo Alto, California, October 1971, pp. 108-116; also in *IEEE Transactions on Communications*, Vol. COM-20, No. 3, Part II, June 1972, pp. 539-546.
21. Metcalfe, R. M., *Packet Communication*, Massachusetts Institute of Technology Project MAC Report MAC TR-114, December 1973.
22. Kleinrock, L. and W. Naylor, "On Measured Behavior of the ARPA Network," *AFIPS Conference Proceedings*, Vol. 43, May 1974, pp. 767-780.
23. Crowther, W. R., F. E. Heart, A. A. McKenzie, J. M. McQuillan, and D. C. Walden, *Network Design Issues*, BBN Report No. 2918. November 1974, to be available from National Technical Information Service.
24. Davies. D. W., and D. L. A. Barber, *Communication Networks for Computers*, London: John Wiley and Sons, 1973.
25. McQuillan, J. M., "The Evolution of Message Processing Techniques in the ARPA Network," to appear in *International Computer State of the Art Report No. 24: Network Systems and Software*, Infotech. Maidenhead, England.
26. Tomlinson, R. S., *Selecting Sequence Numbers*, INWG—Protocol Note #2, August 1974, available as with Reference 9.
27. Cerf, V. and R. Kahn, "A Protocol for Packet Network Intercommunications," *IEEE Transactions on Communications*. Vol. COM-22, No. 5, May 1974, pp. 637-648.
28. Cerf, V., *An Assessment of ARPANET Protocols*, RFC 635, NIC 30489, April 1974, available as with Reference 9.
29. Metcalfe, R. M., "Strategies for Operating Systems in Computer Networks." *Proceedings of the ACM National Conference*, August 1972. pp. 278-281.
30. Retz, D. L., "Operating System Design Considerations for the Packet Switching Environment," these proceedings.
31. Forgie, J. W., "Speech Transmission in Packet-Switched Store-and-Forward Networks," these proceedings.
32. Lam, S. S., and L. Kleinrock, "Dynamic Control Schemes for a Packet Switched Multi-Access Broadcast Channel," these proceedings.
33. Kahn, R. E., "The Organization of Computer Resources into a Packet Radio Network," these proceedings.